Blocks in Space: Intelligent Self-Assembly Using Optimal Control Trajectory Planning

Ella M. Atkins, Gina D. Moylan

University of Maryland, Space Systems Laboratory
382 Technology Drive, College Park, MD, 20740
{ella | gmoylan} @ssl.umd.edu

Abstract

Many different layers of automation must be integrated to support future space missions. At the base layer, spacecraft must autonomously navigate; i.e., follow a specified trajectory using the spacecraft's actuators, sensors and knowledge of its dynamics. This trajectory must minimize precious fuel use and satisfy mission goals and environmental constraints. To facilitate an appropriate connection between task and optimal trajectory planning, we map the well-known Blocks World domain to the space environment by defining a simple task-level implementation that uses cost information from an optimal trajectory planner to make action choices. Our method is applicable at both the micro-level where obstacles must be efficiently circumvented and the macro-level where orbital dynamics dictate assembly task sequencing and trajectory design.

Introduction

Imagining a child stacking blocks on the floor is a pleasant exercise many people can relate to. Placing these same blocks in the space environment and having them self-assemble into particular "stacked" configurations is anything but child's play. When considering the necessary role automation must play in future space missions and endeavors, it is important to study basic scenarios that further our understanding of the challenges we must overcome to meet such objectives. We believe some of these challenges lie in the inherent disconnect between the planning of tasks and the development of the continuous trajectories that must be followed to accomplish these tasks. Before any other mission tasks/goals can be prioritized and fulfilled, spacecraft must be able to autonomously navigate; i.e., follow a specified trajectory using the spacecraft's actuators, sensors and knowledge of its dynamics. To be clear, we make an important distinction between path and trajectory:

- A **path** is the locus of waypoints followed during motion; i.e., a purely geometric motion description.
- A **trajectory** is a path that includes velocities and/or accelerations at each point according to the governing equations of motion; i.e., a geometric and temporal description of the object's motion.

Mission goals are fulfilled by selecting action choices that optimize fuel use and time given system and environmental constraints. While these action choices can be easily implemented with traditional AI planning tools, the optimization of the fuel/time resources required to move through space requires consideration of system dynamics and actuation capabilities involving complex physical motions governed by nonlinear differential equations. Optimal trajectory planners are specifically designed for this task, connecting spatial waypoints with a physically-achievable trajectory. However, they are not designed to optimize over a global assembly problem involving a large number of choices in terms of what gets assembled, when and where (Henshaw, 2003). Full automation—especially in a space environment—must involve the coordination of intelligent task and motion planning.

The problem we address in this paper is set within the context of the famous microworld domain known as "Blocks World" (BW), where an infinite table holds a finite set of unique blocks to be stacked in particular configurations (Slaney and Thiébaux, 2001):

> **Problem Definition:** A group of 4 self-actuating blocks are deployed so that they are in approximately the same orbit but have slightly different positions. From an initial 'snapshot,' the goal is to build a linear 4-block structure in a fuel-optimal 'stacking arrangement' using `block-a` as the anchor block (see Figure 1).

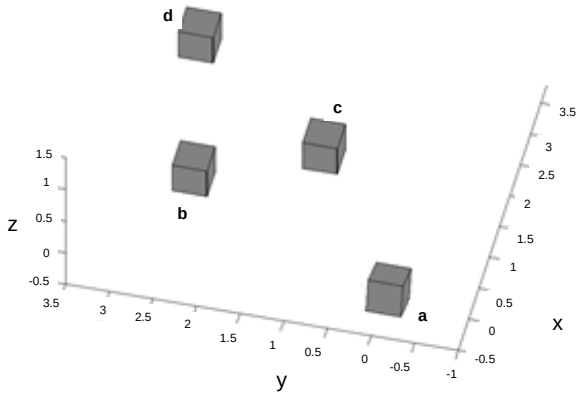


Figure 1: Initial local distribution of blocks.

Typical BW predicates such as `Stack(a,b)` are changed to operators like `Dock(a,b)` and so forth, with the "table" being the orbit in which the configuration resides. In our problem there is no robotic arm moving the blocks about as they are presumed self-actuated by thrusters located on each face.

The merit of considering 'simple' problems like these in terms of integrating sound trajectory planning and task planning components becomes apparent as the action choices of the problem scale (Cambon et al., 2003). It is also fundamental when one considers the many layers of reasoning required to perform even relatively mundane tasks in the complex harsh environment of space. By presenting the AI task planner with appropriate costs in terms of minimizing fuel use, the time-to-completion, and proximity to obstacles (for safety considerations), the trajectory planner relieves the task planner of the full representational details of block locations/motion that would typically overwhelm it, making optimal solutions prohibitive (Smith et al., 2000). Instead, the task planner uses cost values computed by the trajectory planner to influence decisions related to the optimal completion of the block docking sequence.

We lay the groundwork for facilitating a connection between AI task planning and optimal trajectory planning, after a brief discussion of related work, by defining a space-based symbolic domain representation of BW. We then present the system architecture and its component algorithms followed by results for the four-block assembly problem. The results are contrasted with assembly constructions in flat-space and in a central gravitational field (Keplerian model). We conclude with future work to extend our models/algorithms and to practically implement this system for space-based construction activities.

Some Related Work

This work embodies two main themes: self-assembly and the integration of task and optimal trajectory planning. There is a broad range of work encompassing many aspects of the self-assembly mechanics and automation required by this problem (Jones and Matarić, 2003; Suh et al., 2001; Shen, 2001; Butler and Rus, 2001; Shen et al., 2000; Rus and Vona, 1999). Some approaches focus on distributing path-planning and actuation within the system—developing a parallel local awareness (Butler and Rus, 2001), while others focus on global strategies. Both are needed in the space environment where the optimization of limited resources directly impacts mission success. To this end, either strategy requires the careful planning of the trajectories executed in the self-assembly process. One can think of this in other important contexts, such as assembling waypoints to meet military objectives or reconnaissance goals (Petterson and Doherty, 2004).

While there is much work on either problem there remains the persistent gap between the language we use for symbolic reasoning and that used for controlling autonomous motion. This 'gap' is beginning to be recognized and pursued.

Domain Description

BW is a generic domain where the blocks are representations of objects—be they freight, transportation devices, building materials, atoms, etc. The combination of abstraction and the basic premise of moving and assembling these "blocks" in particular configurations lends itself well to the problem of self-assembly.

Mapping BW to a 3D space environment necessitates a paradigm shift in the traditional representation of the 'infinite table.' Instead of a table on which all of the blocks—be they 'free' or part of a tower—reside, we introduce the notion of a 'target table' that will be the orbit in which the blocks are assembled. Blocks in other orbits may be considered free or on 'virtual tables.' Full 3D construction with local and global assembly entails such details as:

- Moving and docking/undocking block superstructures, changing the system dynamics
- Docking in any orbit
- Building structures with 'non-reachable' or variable configurations (e.g., cubic docking with an open center for unique configurations)

Before embracing these details, we have chosen to begin with a constrained construction that disambiguates block states and more closely mirrors the traditional BW paradigm—i.e., the construction of towers or linear assemblies relative to a specific anchor block freely drifting in space. This provides a simplified baseline from which to add the necessary details for unconstrained 3D construction in future work.

PDDL Domain Model

To provide a framework for discussion and to lend some familiarity for those experienced with BW planning, we define a 3D BW domain with a PDDL v.2.1 (Fox and Long, 2003) representation (see Figure 2), making the following assumptions for linear self-assemblies:

1. Blocks have two specific faces (+y,-y) to which any other block may dock.
2. Actions occur sequentially—only one block may be docked/undocked to/from another block at any time.
3. Only one connected structure may be assembled.

```
(define (domain blocks-in-space)
  (:requirements :equality)
  (:predicates
    (block ?b)                 ; ?b is a block
    (orbit ?o)                 ; ?o is an orbit
    (face ?f)                  ; ?f is a block face
    (in-orbit ?b ?o)           ; block ?b is in orbit ?o
    (clear ?b ?f)              ; face ?f of block ?y is clear (undocked)
    (docked ?b1 ?f1 ?b2 ?f2)   ; face ?f1 of block ?b1 is docked to face ?f2 of block ?b2
    (assembled ?b)             ; block ?b is part of the single assembled structure
    (free ?b))                 ; block ?b is free to move and not assembled; both faces of ?b are clear

  (:action insert              ; insert ?block1 residing in ?orbit1 into ?orbit2
     :parameters      (?block1 ?orbit1 ?orbit2)
     :precondition    (and  (block ?block1) (free ?block1) (orbit ?orbit1) (orbit ?orbit2)
                            (in-orbit ?block1 ?orbit1))
     :effect          (and  (in-orbit ?block1 ?orbit2) (not (in-orbit ?block1 ?orbit1))))

  (:action dock               ; move free ?block1 so that its ?face1 docks to ?face2 of anchor ?block2
     :parameters      (?block1 ?face1 ?block2 ?face2)
     :precondition    (and  (block ?block1) (block ?block2) (face ?face1) (face ?face2) (free ?block1)
                            (clear ?block2 ?face2) (assembled ?block2) (not (= ?face1 ?face2)))
     :effect          (and  (docked ?block1 ?face1 ?block2 ?face2) (assembled ?block1)
                        (not (clear ?block1 ?face1)) (not (clear ?block2 ?face2)) (not (free ?block1))))

   (:action undock            ; move ?block1 to undock from anchor ?block2
     :parameters      (?block1 ?face1 ? ?block2 ?neg-face2)
     :precondition    (and  (docked ?block1 ?face1 ?block2 ?face2) (clear ?block1 ?neg-face)))
     :effect          (and  (free ?block1) (clear ?block1 ?face1) (clear ?block2 ?face2)
                          (not (docked ?block1 ?face1 ?block2 ?face2)) (not (assembled ?block1)))))
(define (problem assemble-four-blocks)
  (:domain blocks-in-space)
  (:objects       block-a, block-b, block-c, block-d, pos-y, neg-y, target-orbit)
  (:init         (block block-a) (block block-b) (block block-c) (block block-d) (face neg-y)
                 (face pos-y) (in-orbit block-a target-orbit) (in-orbit block-b target-orbit)
                 (in-orbit block-c target-orbit) (in-orbit block-d target-orbit) (clear block-a pos-y)
                 (clear block-a neg-y) (clear block-b pos-y) (clear block-b neg-y) (clear block-c pos-y)
                 (clear block-c neg-y) (clear block-d pos-y) (clear block-d neg-y)
                 (assembled block-a)  (free block-b) (free block-c) (free block-d))
```

Figure 2: 3D BW problem PDDL representation

4. Blocks either drift as part of the assembly or are free to maneuver, in which case they cannot be connected (docked) to any other block.
5. Spacecraft (blocks) have sufficient fuel for maneuvers and will always execute actions accurately.
6. All blocks are uniquely labeled. They may be interchangeable to allow random placement (as in our example), or they may have specific configuration requirements.

To achieve a goal sequence a set of constructive actions is performed given certain preconditions. When a constructive move is not possible the problem is in a 'deadlocked' state, necessitating the movement of some block before a constructive move is possible. Efficient search strategies employ methods to minimize the number of deadlocks encountered—i.e., backtracking (Slaney and Thiébaux, 2001). In this work, all moves are constructive. However, because optimal solutions are computationally expensive, there is an algorithmic tradeoff of efficiency for optimality due to the nature of the problem as discussed in the following section.

Architecture

In order to facilitate an appropriate link between task and trajectory planning, it was important to design an architecture (see Figure 3) that retained the dynamical state information for each planning state (search node) while keeping this information hidden—i.e., in a "black box"—from the task planner. For this initial implementation, a primitive C++ "task planner" interprets the PDDL domain (see Figure 2) and conducts an optimal search in which the "translator function" is invoked to acquire the cost J of the instantiated action by interfacing with a Matlab-based optimal trajectory planner (Henshaw, 2003). The task planner uses a uniform cost (Dijkstra's) search strategy with actual node n cost, $g(n)$, set to the cost of the parent node plus additional cost J of transitioning from the parent to node n. Transition costs J (see Equation 2) are computed during the trajectory planning process for each action. The focus of the current task planner's

implementation is the information communicated between task and trajectory planners. We therefore discuss this communication and the design of the trajectory planner in more depth below.

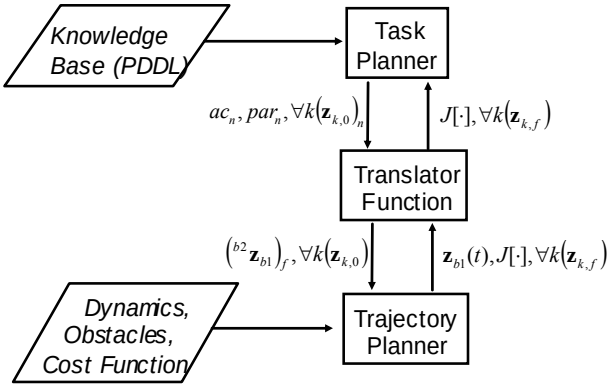


Figure 3: System Architecture

Task Planner and Translator Function

The task planner begins its search for an optimal block assembly sequence by using the knowledge base's initial full-state 'snapshot' $\mathbf{z}_{k,o}$ for all blocks k of the recently deployed system. A node queue is then constructed, with children of the initial root node formed from actions ac_n and parameter bindings par_n that are able to preferentially achieve a subgoal or the set of actions (ac_n,par_n) that meet all preconditions of ac_n if no subgoals can be directly achieved. For each uniform cost node expansion the queue must be ordered by total path cost, $g(n)$.

The translator function is then called to compute the cost J of transitioning from each parent to child node. The translator passes the continuous state $\mathbf{z}_{k,o}$ for all blocks directly to the trajectory planner. Based on the $\mathbf{z}_{k,o}$ and the action (ac_n,par_n) to be executed, the translator also computes the final (goal) state $({}^{b2}\mathbf{z}_{b1})_f$ to be achieved as the product of executing action ac_n. For all BW actions, *b1* is the block to be moved and *b2* is the anchor block or target orbit to which *b1* must maneuver. By expressing position vector $({}^{b2}\mathbf{z}_{b1})_f$ in *b2* coordinates (indicated by the leading *b2* superscript), the translator computes *b1* goal positions relative to its drifting target (*b2*), allowing maneuver time to be optimized by the trajectory planner rather than specified in advance. In the general case where all blocks drift over time (freely or in a gravitational field), all blocks will move as the trajectory for each ac_n is executed, requiring that the final state $\mathbf{z}_{k,f}$ of all blocks after executing ac_n be stored as the initial state "snapshot" for the offspring of node n. Should node *n* be part of the optimal solution, the optimal trajectory $\mathbf{z}_{b1}(t)$ for maneuvering block *b1* is archived along with the cost J returned by the translator as the cost of executing action (ac_n,par_n).

This might appear a lengthy computational process for such a simple assembly activity (Cambon et al, 2003). However, as we discuss below, incorporating complex dynamics is the only way to ensure optimal solutions worthy of space applications.

Trajectory Planner

Traditional trajectory planning strategies used in terrestrial applications for rover locomotion, etc., need to be supplemented to accommodate the unique challenges of navigating spacecraft. To navigate a *free path* from point *A* to point *B* without intersecting any obstacles, cell decomposition and roadmap methods such as Voronoi diagrams, etc. are usually employed with varying degrees of success (Latombe, 1991). However, when dealing with spacecraft one must take into account:

- Limited fuel resources/maneuverability,
- Possible encounter(s) with a wide range of obstacle operating velocities,
- Dynamic thruster constraints,
- Varied endpoint constraints/operating times.

Traditional methods fall short of meeting these dynamic constraints (Henshaw, 2003). This is especially realized when contrasting a relative 'straight-line' trajectory in flat space with the same trajectory in a central gravitational field as presented below. Additionally, the self-assembly of blocks in both environments requires the transfer of blocks from one orbit to another. For circular orbits, the optimal global planning strategy for maneuvering blocks to a target orbit is a straightforward Hohmann transfer (Miele et al., 2004).

To fully address these challenges we chose a trajectory planning algorithm specifically designed to solve end-to-end orbital docking problems involving both orbital maneuvering and proximity operations using realistic saturating thrusters (Henshaw, 2003). Robust numerical methods and the use of Calculus of Variations allows the planner to develop a cost functional that penalizes fuel use, obstacle clearance distance, and arrival time while enforcing dynamic orbital constraints. Six degree of freedom paths are found by deriving Euler-Lagrange equations corresponding to the cost functional, then solving the associated boundary value problem using collocation (implemented with the Levenberg-Marquardt algorithm) and continuation techniques, allowing for the optimization of arrival time and fuel use. To avoid error effects, a feedback control algorithm was implemented (using Pontryagin's minimum principle).

The 6-DOF dynamic equations for the moving vehicle are:

$$\dot{\mathbf{z}}[(t)] = \begin{bmatrix} \dot{\mathbf{v}} \\ \dot{\mathbf{p}} \\ \dot{\omega} \\ \dot{\sigma} \end{bmatrix} = \begin{bmatrix} -\mu\mathbf{p}/\|\mathbf{p}\|^3 \\ \mathbf{v} \\ -\mathbf{H}^{-1}S(\mathbf{H}\omega)\omega \\ \mathbf{G}_\sigma(\sigma)\omega \end{bmatrix} + \begin{bmatrix} \mathbf{R}(\sigma)\mathbf{u}(t)/m \\ 0 \\ \mathbf{H}^{-1}\tau(t) \\ 0 \end{bmatrix} \qquad (1)$$

where:

p is vehicle position relative to the anchor/target[1]

v is vehicle translational velocity relative to the target

σ is a modified Rodrigues vector (a 3-element vector representing vehicle attitude without singularities)

ω is the rotational rate vector in the body frame

m is vehicle mass,

H is the rotational inertial matrix,

$R(\sigma)$ is a rotation matrix that converts body to inertial coordinates,

S the matrix representation of cross product $\mathbf{H} \times \omega$

$\mathbf{G}_\sigma(\sigma)$ is the dynamic equation for the Rodrigues vector

$\mathbf{u}(t)$ is the force vector produced by saturating thrusters

$\tau(t)$ is the limited torque vector.

To generate the desired trajectory, a cost functional is minimized subject to the dynamic constraint, $\dot{\mathbf{z}}(t)$:

$$J[\cdot] = \int_{t_o}^{t_f} \left(L_{control}[\mathbf{z}, v, \tau] + L_{obstacle}[\mathbf{z}] + L_{time} + \lambda^T f(\mathbf{z}, v, \tau) \right) dt \qquad (2)$$

where:

$f(\mathbf{z},v,\tau)$ represents vehicle dynamics from Eq. (1)

$L_{control}[\mathbf{z},v,\tau]$ penalizes control effort—fuel use

$L_{obstacle}[\mathbf{z}]$ penalizes obstacle clearance distance

L_{time} penalizes completion time

Finally, the boundary conditions, specified by the Translator (see Figure 3), are defined by the initial state in each search node and the final waypoint computed from the initial state and choice of task to execute. Although the translator need not specify time of arrival at the final state, motion of the target must be known as a function of arrival time which is reasonable given the target vehicle's natural orbital motion. The BW domain as defined for this work specifies trajectory planning problems with fixed arrival locations and either fixed or free arrival times, allowing a relatively simple form of the transversality boundary condition $L_\alpha(\cdot) = 0$ to hold.

4-block Assembly Results

The *4*-block linear assembly problem defined in Figure 2 was cast in a circular equatorial orbit (`target-orbit`) with an orbital radius of 6767.06km (388.92km above the Earth). All blocks/modules had edge length 0.2 km, and the faces were presumed tangent (no separation) when docked.[2] Table 1 lists each block's initial position $\mathbf{p}_i(0)$ relative to anchor `block-a` and inertial orientation $\mathbf{R}(\sigma)$. Initial block relative velocities and angular velocities/accelerations were assumed zero since all blocks approximately occupy the same orbit.

Table 1: Initial block locations

Block-*i*	$\mathbf{p}_i(0)$ (km)	$\mathbf{R}(\sigma)$
`block-a`	(0.0, 0.0, 0.0)	(0.0, 0.0, 1.0)
`block-b`	(1.0, 2.5, 1.0)	(0.0, 0.0, 1.0)
`block-c`	(2.0, 1.5, 0.0)	(0.0, 0.0, 1.0)
`block-d`	(3.0, 3.0, 1.5)	(0.0, 0.0, 1.0)

Before examining the full 4-block planning process, we motivate the use of the full-state trajectory planner by examining a single action: `(dock block-d neg-y block-a pos-y)` with only the `block-d` and `block-a` positions shown in Table 1—i.e., no other blocks acting as obstacles. The optimal trajectory and cost were compared for the dock conducted in flat-space (no gravity) where a simpler path planner might suffice, with the same problem in the circular orbit specified above.

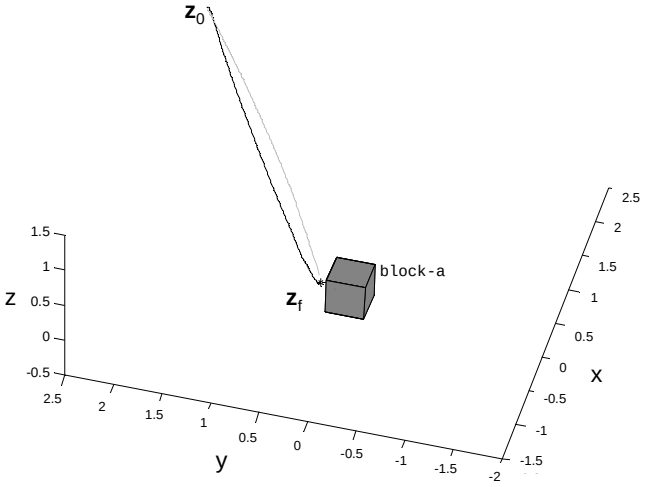
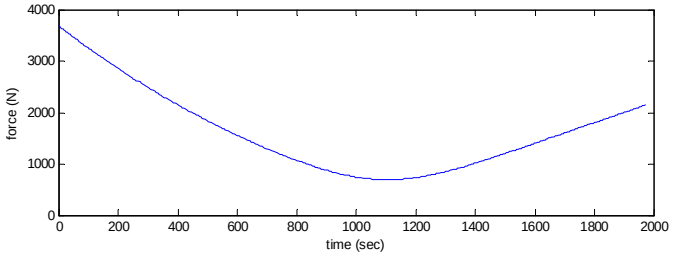


Figure 4: Paths with gravity (black) versus flat-space (grey) models.

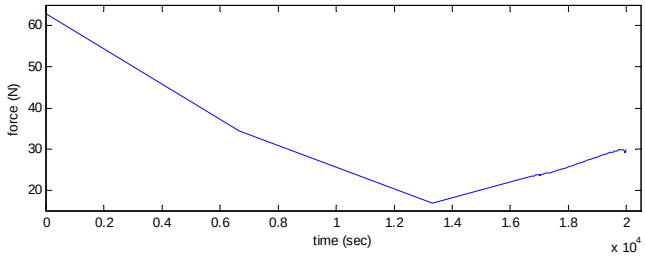
A comparison of paths and force profiles for the two cases is shown in Figure 5. Although the physical paths through space do not differ greatly, there is substantial difference in cost. As shown in Figure 4, the force at any given time is nearly two orders of magnitude higher for the solution with gravity, however, this difference is mitigated to some extent by the reduced time to dock, thereby also lowering the time over which gravitational forces act on the block. Without gravity, the range of single docking operations for the four blocks with initial states given by Table 1 exhibited minimal difference in assembly cost. Conversely, tasks executed with the gravitational model had cost that varied by up to 80% for different docking tasks.

1 The docking target block (or target orbit) may be in motion, but this motion must be modeled a priori—a reasonable assumption for insertion into a known orbit or docking to a controlled spacecraft.

2 Each "block" has rather enormous dimensions for our example. Such size and separation distance values were chosen to simultaneously illustrate the effects of obstacle avoidance and orbital dynamics on cost without modeling significantly more than four blocks.



a) Force (magnitude) vs. time with gravitational effects



b) Force (magnitude) vs. time with no gravity (flat space)

Figure 5: Comparative force plots for `(dock d -y a -y)`

Next we ran the planner (search engine) with the blocks in their circular orbit. A representation of the full search-space is shown in Figure 5 along with the total cost for a number of complete paths through the search space. This 4-block example had branching factors 6, 4, and 2 at search levels 1, 2, and 3, respectively, representing the possible combinations of `dock` operations each unassembled block could achieve. The optimal plan was computed to be: 1) `(dock block-b neg-y block-a pos-y)` (cost J=4.67E3), 2) `(dock block-c neg-y block-b pos-y)` (J=5.91E3), then 3) `(dock block-d neg-y block-c pos-y)` (J=2.29E4), with total assembly cost g=3.35E4. Note that not all Level 2 or 3 nodes were actually expanded with the uniform cost engine; select cost data is provided to illustrate the search-space and facilitate assembly structure and cost comparison.

For non-optimal assemblies, there was a significant range of individual docking task costs: a minimum = 4.67E3 (part of the optimal plan), a maximum = 7.78E4, and an average = 2.25E4. Further, the same final assembly structure does not ensure consistent cost (e.g., assembly **c-a-b-d** in Figure 4). As discussed in Section 3.2, the trajectory planner may return a different cost for the same final assembly based on obstacle avoidance requirements, illustrating the importance of task ordering choices for an optimal assembly.

The layout of this 4-block problem was chosen to demonstrate the functionality of our system and to clearly illustrate the need for the integration of task and trajectory planning. However, this integration comes at the cost of computational time; not uncommon to systems needing any degree of trajectory or even path planning fidelity (Pettersson and Doherty, 2004). The worst runs need over an hour to compute, becoming cumbersome, or even prohibitive, as the number of assembly sequences scales with the number of blocks. Even with this computational burden, there are ways to mitigate the necessity of running all assembly possibilities in the trajectory planner. Making use of selective cost information, combining global and local assembly strategies, and incorporating pre-processing that uses system dynamics to deliver intelligent, best guess cost estimates are some of the ways the system can work in both offline and online capacities.

Level 1 (complete)**:**	**Level 2** (partial)**:**	**Level 3** (partial)**:**	**Final structure**	**Total assembly cost**
1 = (dock b -x a +x)	***1-1 = (dock c +x a -x)***	1-1a = (dock d -x b +x)	c-**a**-b-d	4.25E4
		1-1b = (dock d +x c -x)	d-c-**a**-b	1.025E5
	1-2 = (dock d +x a -x)			
	1-3 = (dock c -x b +x)			
	1-4 = (dock d -x b +x)	1-4a = (dock c +x a -x)	c-**a**-b-d	5.03E4
		1-4b = (dock c -x d +x)	**a**-b-d-c	4.27E4
2 = (dock c -x a +x)				
3 = (dock d -x a +x)				
4 = (dock b +x a -x)				
5 = (dock c +x a -x)				
6 = (dock d +x a -x)	***6-1 = (dock b -x a +x)***	6-1a = (dock c -x b +x)	d-**a**-b-c	5.96E4
		6-1b = (dock c +x d -x)	c-d-**a**-b	7.37E4
	6-2 = (dock c -x a +x)			
	6-3 = (dock b +x d -x)			
	6-4 = (dock c +x d -x)	6-4a = (dock b -x a +x)	c-d-**a**-b	7.37E4
		6-4b = (dock b +x c -x)	b-c-d-**a**	9.47E4
OPTIMAL SOLUTION:				
4 = (dock b +x a –x)	***4-3 = (dock c +x b -x)***	4-3b = (dock d +x c -x)	d-c-b-a	**3.35E4**

Figure 5: Four-block linear assembly search-space and assembly costs

Conclusions and Future Work

Intelligent in-space self-assembly requires careful integration of task-level and physics-based reasoning systems. We have argued that assembly task planning necessarily requires the incorporation and knowledge of complex system dynamics in the overall task planning/sequencing strategy. A framework for establishing a connection between an optimal control trajectory planner and task planner in this domain was established, from which we presented a simple assembly problem that clearly illustrated the need for a trajectory planner designed to work with continuous dynamical system models of in-space assembly problems as well as a task planner to propose action sequences that will successfully achieve assembly goals.

This work began with planning algorithms and a PDDL model designed for a small set of blocks and sequential assembly choices due to the computationally-intensive planning processes involved. Its primary contribution lies in the integration of optimal task and trajectory planners, specifically the knowledge representation and interface language that enable the task planner to manage complex trajectories while processing only a small set of symbolic features and a single measure of cost for each planning state. A secondary contribution is the PDDL 3D BW domain definition (Figure 2). Although a sophisticated trajectory planner is already in place, a more capable task planner will be required to increase search efficiency and enable parallel, multi-tasked activity schedules—an important capability when many "blocks" are assembled.

Our aim for future work is to improve algorithmic efficiency by utilizing (admissible) heuristics to reduce search-space size while maintaining optimality and expanding the task planner to handle parallel assembly task execution. The architecture and BW representation presented builds a foundation on which both AI and control systems researchers can build such extensions.

References

A. Miele, M. Ciarcià, J. Mathwig. 2004. "Reflections on the Hohmann Transfer," Journal of Optimization Theory and Applications Vol. 123, Issue 2, pp. 233 – 253.

Stéphane Cambon, Fabien Gravot and Rachid Alami. 2004. "A Robot Task Planner that Merges Symbolic and Geometric Reasoning," Proceedings of the 16th Annual Conference on Artificial Intelligence, Spain.

Pettersson, P-O., Doherty, P. 2004. "Probabilistic Roadmap Based Path Planning for Autonomous Unmanned Aerial Vehicles," Proceedings of the 14th International Conference on Automated Planning and Scheduling.

Chris Jones and Maja J. Matarić. 2003. "From Local to Global Behavior in Intelligent Self-Assembly," Proceedings of the IEEE International Conference on Robotics and Automation.

R. Lampariello, S. Agrawal, G. Hirzinger. 2003. "Optimal Motion Planning for Free-Flying Robots," International Conference on Robotics and Automation, Taiwan.

Fox, M. and Long, D. 2003. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains," Journal of Artificial Intelligence Research 20, pp. 61-124.

Henshaw, C.G. 2003. "A Variational Technique for Spacecraft Trajectory Planning," PhD Dissertation: Department of Aerospace Engineering, University of Maryland College Park.

John W. Suh, Samuel B. Homans and Mark Yim. 2002. "Telecubes: Mechanical Design of a Module for Self-Reconfigurable Robotics," Proceedings of the 2002 IEEE International Conference on Robotics and Automation, Washington DC.

Jacobsen, S., Lee, C., Zhu, C., and Dubowsky, S. 2002. "Planning of Safe Kinematic Trajectories for Free Flying Robots Approaching an Uncontrolled Spinning Satellite" Proceedings of the ASME 27th Annual Biennial Mechanisms and Robotics Conference, Montreal.

John Slaney and Sylvie Thiébaux, "Blocks World Revisited," Artificial Intelligence, 125, pp. 119-153, 2001.

Wei-Min Shen. 2001. "Metamorphic Robotic Systems for Space Exploration," ICASE/USRA/LaRC Workshop On Revolutionary Aerospace Systems Concepts for Human/Robotic Exploration of the Solar System.

Zack Butler and Daniela Rus. 2001. "Distributed motion planning for 3D modular robots with unit-compressible modules," Proceedings of the Internnational Conference on Intelligent Robots and Systems.

Wei-Min Shen , Yimin Lu , Peter Will. 2000. "Hormone-based control for self-reconfigurable robots," Proceedings of the Fourth International Conference on Autonomous Agents, Barcelona.

David E. Smith, Jeremy Frank, and Ari Jónsson. 2000. "Bridging the Gap Between Planning and Scheduling," Knowledge Engineering Review 15(1).

Daniela Rus & Marsette Vona. 1999. Self-reconfiguration Planning with Compressible Unit Modules," Proceedings of the 1999 IEEE International Conference on Robotics and Automation.

J. Scott Penberthy and Daniel S.Weld. 1994. "Temporal Planning with Continuous Change," AAAI.

Jean-Claude Latombe. 1991. Robot Motion Planning, Kluwer.